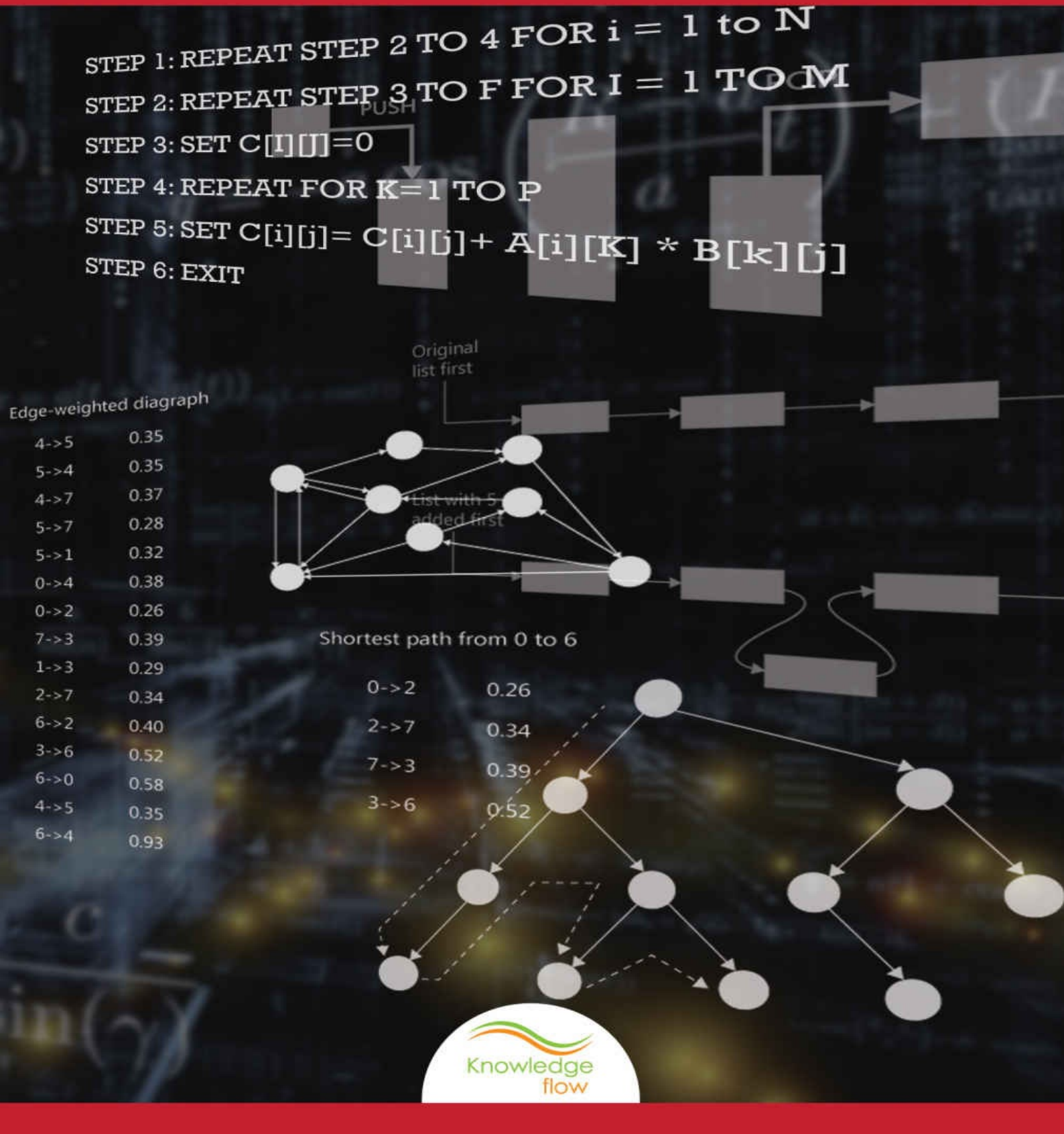
Knowledge flow

DATA STRUCTURES AND ALGORITHMS



Knowledge flow

“LEARNING STARTS WITH VIEWING THE WORLD DIFFERENTLY.”

Knowledge flow- A mobile learning platform provides apps, eBooks and video tutorials

Knowledge flow brings you learning eBook of ***Data Structures and Algorithms***. This eBook is for all information technology and computer science students and professionals across the world.

Follow us on

Facebook

Google plus

Twitter

For more information visit us at

Knowledgeflow.in

knowledgeflowapps.blogspot.in

Thank you for using Knowledge flow eBooks

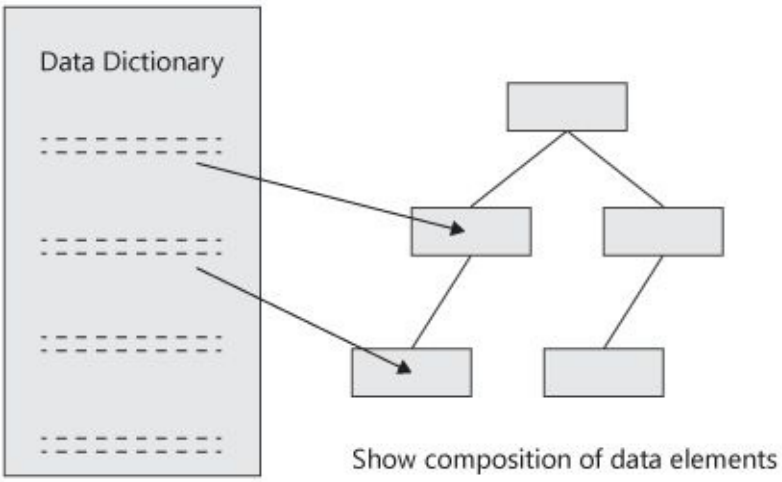
DATA STRUCTURES AND ALGORITHMS

Disclaimer

This eBook contents is for informational and study purposes only. The ***Knowledge flow*** makes no claims, promises, or guarantees about the accuracy, completeness, or adequacy of the contents of this eBook and no legal liability or other responsibility is accepted by ***Knowledge flow*** for any errors, omissions, or statements on this eBook.

Introduction

- Data Structures is a study or a technique used to understand how to store a bunch of data in an organized manner, so it can be used in a very sophisticated manner for designing of programs and algorithms.



Data structure diagram

- Data structure is a vital subject to the discipline of Information and Computer technology.
- In simple words Data structure is nothing but storing and then fetching of the data from any part of the memory of the computer in an organized manner whenever needed.
- The term Data Structure was first coined into existence in year 1976 written by Niklaus Wirth in his book "*Algorithms + Data structures = Programs*", which covered some fundamental details of computer programming.

Types of data structures

- *Arrays:* Simplest data structure which shows linearity as a property. The elements are arranged in a linear manner. Let us choose X as a name for the array. And the elements are referenced respectively by numbers 1,2,3,….,n .Therefore it can be denoted by

1. Subscript notation: $x_1, x_2, x_3, \ldots\ldots\ldots, x_n$;
2. By parenthesis notation: K(1), K(2), K(3), …….,K(N);
3. By bracket notation: K[1], K[2], K[3], ……, K[N].

Now Y in K[Y] is called subscript and K[Y] in whole is called a subscripted variable.

- *Linked Lists:* In this type of data structure it's difficult to fetch data from the data since it might be massive. In this particular situation we use the tools like pointer

and link. Both pointer and link have different functionality.

- *Pointer:* is used when it's needed for an element in one list to point to an element in a different list. Link is used when an element in a list points to an element in that same list.
- *Trees:* This type represents the whole data present in a hierarchical form on the basis of the relationship between the elements in that group of data.
- *Stack:* It's last in first out. In this type of data structure the data are inserted and deleted from one point. It's just one way in and out.
- *Queue:* It is first in first out. In this data is inserted from one side which will be the front and pushed to the other side till it reaches the end that is the rear side. It has two ways, one way in and another way out.
- *Graph:* It depicts relationship between pairs of elements.

Operations of data structures

- *Traversing:* Accessing or visiting the record to be processed.
- *Searching:* Finding the record or the location of the record with the key value.
- *Insertion:* Adding a record.
- *Deletion:* Removal of record.
- *Sorting:* Arranging records in the needed order.
- *Merging:* Combining two records into a single file.

Data Structure for Strings

Strings in data structure are character data in sequence.

Characters used are:

Alphabet: A, B, C,…., Z.

Digit: 0 1 2 ….. 9.

Special characters are: + - / * ( ) , . $ = ‘

Storing of strings

- String can be stored in three types of structures:
- Fixed-length structures,
- Variable-length structures with maximum limit set,
- Three linked structures.

Fixed-length storage

In this each line is seen as a record and all records have the same length.

Main advantages of this way of storing strings are:

- Easy to access data.
- Easy to update data.

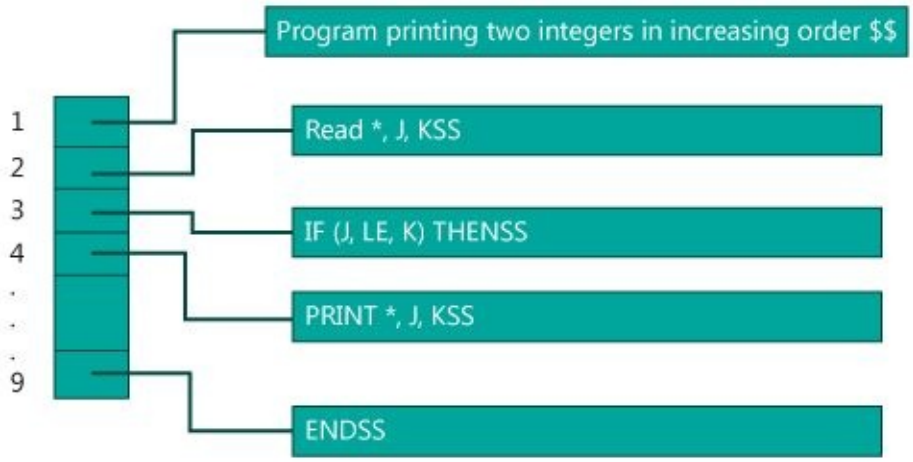
Main disadvantages of this kind of storing technique:

- Waste of time due to inessential blank space.
- Changing even a misspelled word leads to change in the whole record.

Variable-length structure with maximum limit set

The storage can be done in two general ways:

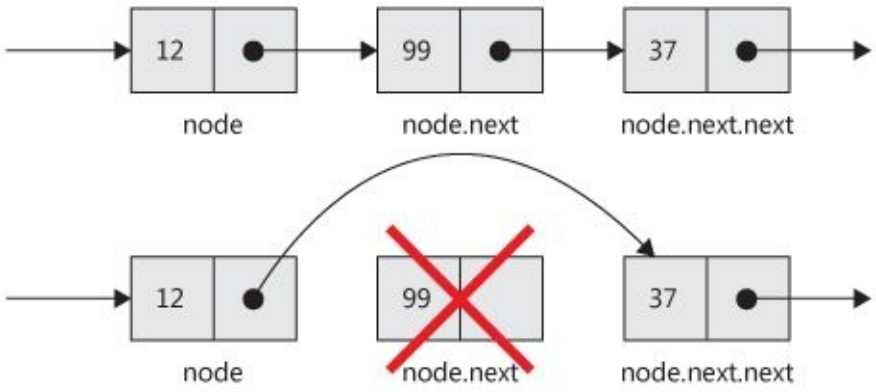
- Use of a marker to show the end of the string.
- Length of the string can be listed as an item in the pointer array.



Record with sentinels

Linked storage

In this memory cells are assigned to a character and also a link that would provide them the address of the cell that has the next character in the string.



Linked list

Operations of string and its data type

String oriented operations are.

- *Substring:* To access a substring there is need of three key values:
- String name,
- Position of substring in the string,
- Substring length.
- *Indexing:* This is done by pattern matching.
- *Concatenation:* It is dented by //. For example *P* and *B* are two strings. *P//B* states that character of *P* is followed by that of *B*.
- *Length:* Total number of character in a string depicts the length.

Word processing

Word processing just modifies the data.

The operations of word processing are as follows:

- *Replace operation:* Replacing the given string with another.
- REPLACE (text, pattern a, pattern b)
- *Insertion:* To add or insert a string.
- INSERT (text, location, string)
- *Delete operation:* Deletion of string.
- DELETE (A, INDEX (A, B), LENGTH (B)).

Pattern matching algorithms

Pattern matching is the complexity that arises to find out whether the given string pattern A appears in a string text D.

Note: Sometimes in pattern matching algorithms, characters are denoted in lower case letters format such as x, y, z …..and for denoting repetition exponents are used.

For example

$x^2 y^3 xy^3$ for xxyyyxyyy and $(pq)^3$ for pqpqpq

Empty string can be denoted by Greek letter lambda (ʎ).

First Pattern Matching Algorithm

In this kind of pattern matching algorithm we compare pattern A with each substring of pattern D. Therefore moving the match process is done from left to right till it gets a match to that of the pattern.

Algorithm:

A and D are string with length X and Y, respectively, where they are stored as arrays with one character per element. Find out the index of A in D.

STEP 1: Set B ≔ 1 and MAX ≔ X – Y + 1.

STEP 2: REPEAT THE STEP 3 TO 5 while B ≤ MAX:

STEP 3: REPEAT FOR Q=1 to X:

If A [Q]≠ D[B + Q – 1], THEN:

STEP 4: Set INDEX = B, AND EXIT. //Success//

STEP 5: Set B ≔ B + 1. //End of step 2 outer loop//

STEP 6: Set INDEX = 0.

STEP 7: Exit.

Note: Complexity in this pattern states that the time required executing the algorithm is n^2. Where n = x + y, when A is x - character string and B is y – character string.

Second pattern matching algorithm

This pattern matching algorithm does use a table which is derived from particular pattern A but independent of the text D.

Let's consider,

A = aaaba

Then the D has one of the following three forms:

- D = aaab…
- D = aaa….
- D = aax…

Algorithm:

The pattern matching table $F(W_{1,} H)$ of a pattern A is in memory, and the input is an Z – character string $H= H_1 H_2 \ldots\ldots H_{n.}$ This algorithm finds the INDEX of A in H.

STEP 1: Set B ≔ 1 and $Y_1 = Q_0$

STEP 2: Repeat steps 3 to 5 while $Y_k \neq A$ and $B \leq Z$.

STEP 3: Read $H_{k.}$

STEP 4: Set $Y_{k+1} ≔ F (Y_k + H_k)$.

STEP 5: Set B ≔ B + 1.

STEP 6: If $Y_K = A$, then:

INDEX = B – LENGTH (A).

Else:

INDEX = 0.

STEP 7: Exit.

Note: In this the pattern matching complexity is equal to n when compared to that of first pattern matching algorithm.

Arrays and Pointers

Array represents the linear data structure with sequential memory allocations.

Linear arrays

It is a list of a finite number of same kinds of data element:

- Elements are stored in successive memory allocation.
- Elements are referenced by index.

Traversing of linear arrays

By this we can carry out a particular process over the particular group of elements. Suppose we want count particular kind of element and then print them. This can be done using traversing a linear array.

Algorithm:

This algorithm traverses a linear array A with the upper bound U and lower bound L.

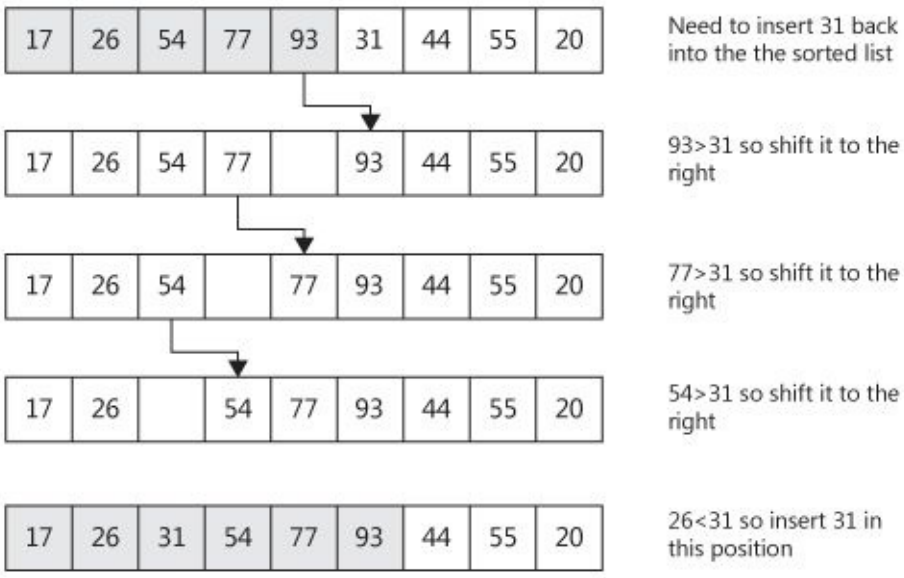
STEP 1: Repeat for B ≔ L to U.

Applying process to A[B].

STEP 2: Exit.

Inserting and Deleting

- It states the operation of inserting *i.e.* adding and deleting *i.e.* removal of element.
- Deleting and inserting an element at the end is not difficult or complex but inserting and deleting in middle can be complex because of the linear property of the array.



Insertion pass

Algorithm:

Deletion from array Delete (A, M, K, Item)

STEP 1: Set item = A[K]

STEP 2: REPEAT FOR j = K TO M-1

STEP 3: SET A[j]= A[J+1]

STEP 4: SET N = N-1

STEP 5: EXIT

Matrix multiplication(A, B, M, C, M, N,P)

STEP 1: REPEAT STEP 2 TO 4 FOR i = 1 to N

STEP 2: REPEAT STEP 3 TO F FOR I = 1 TO M

STEP 3: SET C[I][J]=0

STEP 4: REPEAT FOR K=1 TO P

STEP 5: SET C[i][j]= C[i][j]+ A[i][K] * B[k][j]

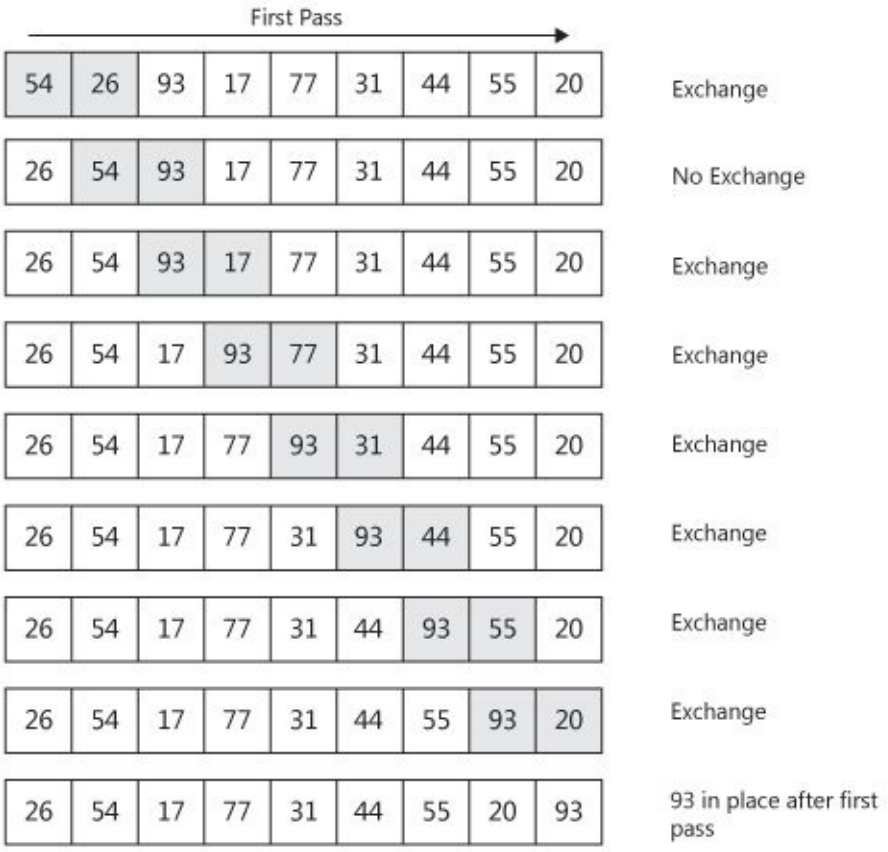
STEP 6: EXIT

Sorting

Here sorting is to rearrange the elements in such a manner so that they are in increasing order. For example the given numbers are 8, 4, 5, 7, and 1.

Therefore after sorting we get 1, 4, 5, 7, and 8.

Sorting has a technique involved known as bubble sort.



Bubble sorting

Procedure for bubble sort:

- Compares and arranges them in desired order.
- P [M-1] with P [M] and arrange them so that P [M-1] < P [M].
- Repeat above procedure with one less comparison. Possibly rearrange P [M-2] with P [M-1].
- Now repeat the first procedure with two fewer comparisons. Possibly rearrange P [M-3] and P [M-2].
- After step [M-1] the list will be sorted in increasing order.

Searching in data structure

- Linear search
- Binary search

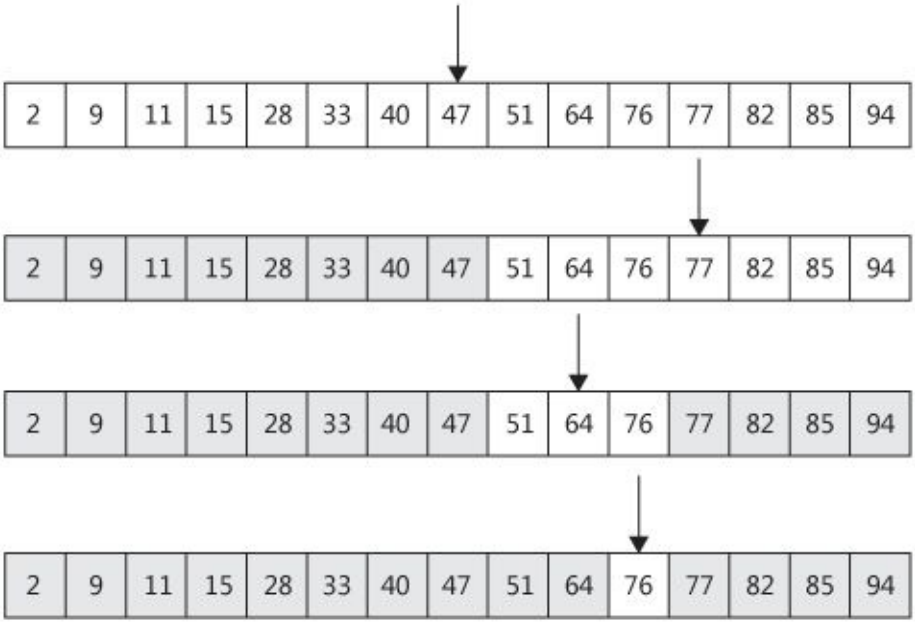
Linear search

It’s a sequential search which goes from one end to all the way to the last end till it doesn’t end up finding the data that is needed or the value is for search.

Binary search

Therefore the binary search opens the directory in the middle to determine which half

contains the name being sought and that particular half is also divided into two to see which part contains the value being sought.



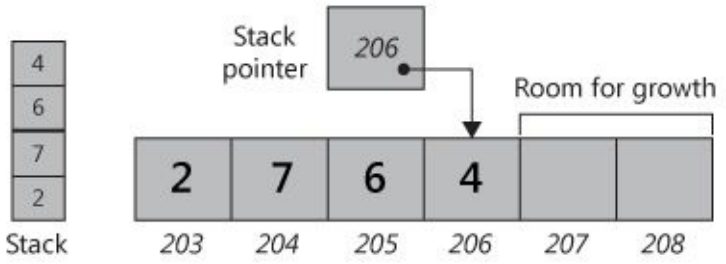
Binary search

Multidimensional Arrays

This array has more than one-dimensional arrays i.e. more than one subscript references the elements of array.

Pointers Arrays

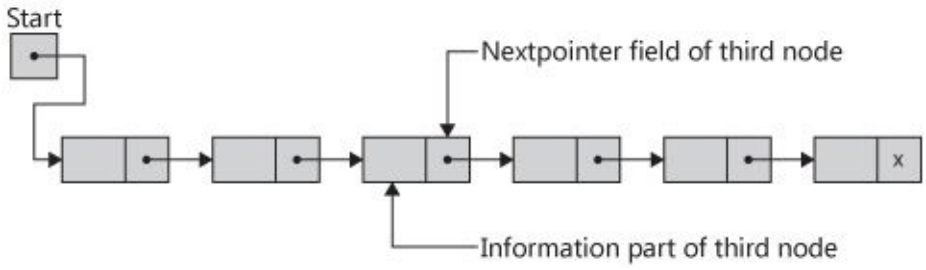
Modification of two space-efficient data structure can be easily done for the indexing of the individual group. For this particular process pointer array can be used. We have a value as array. A variable X as a pointer and if it points out an element in the value and if B has the address of an element in value. Then the variable B is called a pointer.



Pointer array

Linked Lists

A linked list is a linear sequenced collection of data elements which s named as nodes and theses are given a linear arrangement with the help of pointers.



Linked list

Traversing a linked list

Traversing is done using a pointer.

Algorithm:

STEP 1: SET POINTER ≔ START.

STEP 2: Repeat step 3 and 4 while POINTER≠ NULL.

STEP 3: Apply PROCESS to INFO [POINTER].

STEP 4: Set PTR ≔ LINK [POINTER]. //POINTER POINTS TO NEXT NODE//

STEP 5: EXIT.

Searching a linked list

There two conditions while we are searching a list. It totally depends on the nature of the list which would determine the time taken for the particular list location to be known.

The conditions are.

- If the list is unsorted.
- If list is sorted.

Algorithm for the list if unsorted:

STEP 1: Set POINTER ≔ START.

STEP 2: Repeat Step 3 while POINTER ≠ NULL.

STEP 3: If ITEM = INFO[POINTER], Then:

Set LOC ≔ POINTER, and Exit.

Else:

Set POINTER ≔ LINK [POINTER].

STEP 4: Set LOC ≔ NULL.

STEP 5: EXIT.

Algorithm for the list if sorted:

STEP 1: Set POINTER ≔ START.

STEP 2: Repeat Step 3 while POINTER ≠ NULL.

STEP 3: If ITEM < INFO [POINTER], Then:

Set POINTER ≔ LINK [POINTER].

Else if ITEM = INFO [POINTER], Then:

Set LOC ≔ POINTER, and Exit.

Else:

Set LOC ≔ NULL, and Exit.

STEP 4: Set LOC ≔ NULL.

STEP 5: EXIT.

Insertion into a linked list

Three ways to insert in the list:

Inserting at the beginning of a list-

Algorithm:

STEP 1: If Avail = Null, then: Write: NO SPACE and Exit,

STEP 2: Set New ≔ Avail and Avail ≔ LINK [Avail].

STEP 3: Set INFO [New] ≔ DATA.

STEP 4: Set LINK [New] ≔ Start.

STEP 5: START ≔ New.

STEP 6: EXIT.

Inserting at the given node-

Algorithm:

STEP 1: If Avail = Null, then: Write: NO SPACE and Exit,

STEP 2: Set New ≔ Avail and Avail ≔ LINK [Avail].

STEP 3: Set INFO [New] ≔ DATA.

STEP 4: If LOC = NULL, THEN:

Set LINK [New] ≔ Start and START ≔ New.

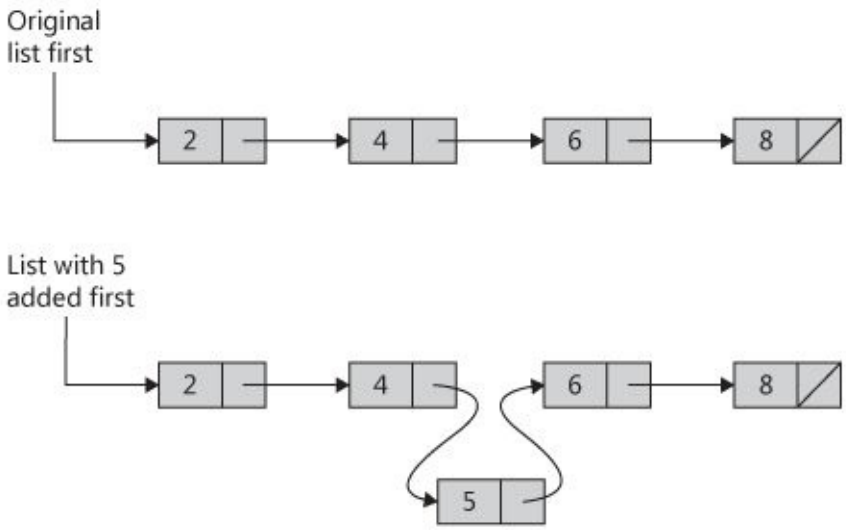
Else: Set LINK [New] ≔ LINK [LOC] and LINK [LOC] ≔ New.

STEP 6: EXIT.

Inserted into a sorted linked list-

When item is to be inserted into the sorted linked list, then item has to be inserted between two nodes.

INFO(X) < ITEM < INFO(Y)



Insertion of data into the linked list

Deletion of linked list

Procedure of deletion from linked list:

- The next pointer fields of node X now points to node Y. therefore before X pointed to node M which further pointed towards the node Y.
- The next pointer M now points to the original first node in the free pool.
- AVAIL now points to the deleted node M.

Header linked list

Linked list containing a header node, which is positioned at the beginning of the list is a special node is called header linked list.

Two header linked list that are most commonly used are.

- Grounded header list: This header contains the last node with null pointer.
- Circular header list: This header contains the last node which points back to the header node.

Stacks and Queues

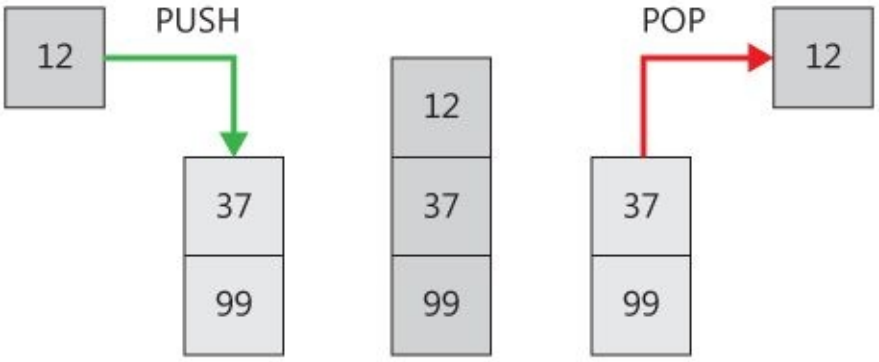
Sometime the user want to stop the freedom of insertion and deletion from any part of the list to maintain linearity and regularity therefore at that moment or situation stacks and queues are needed. It follows LIFO which states last in first out.

Stacks

Stack is a list which permits the user to only insert or delete the elements in the list from one end, know as top of the stack.

Two basis operations of stack are.

- *Push:* Term to insert element.
- *Pop:* Term used for removal of the element.



Stack

Algorithm for PUSH:

STEP 1: If Top = Max, Print: NA, and Return,

STEP 2: Set Top ≔ Top + 1.

STEP 3: Set STACK [TOP] ≔ ITEM.

STEP 4: RETURN.

Algorithm for POP:

STEP 1: If Top = 0, Print: NA, and Return,

STEP 2: Set ITEM ≔ STACK [TOP]

STEP 3: Set Top ≔ Top - 1.

STEP 4: RETURN.

Note: Term minimize flow states the difference between underflow and overflow.

Recursion

It is the repetition or calling back of the statement itself. For a statement to be referred as recursive statement it should have following two properties:

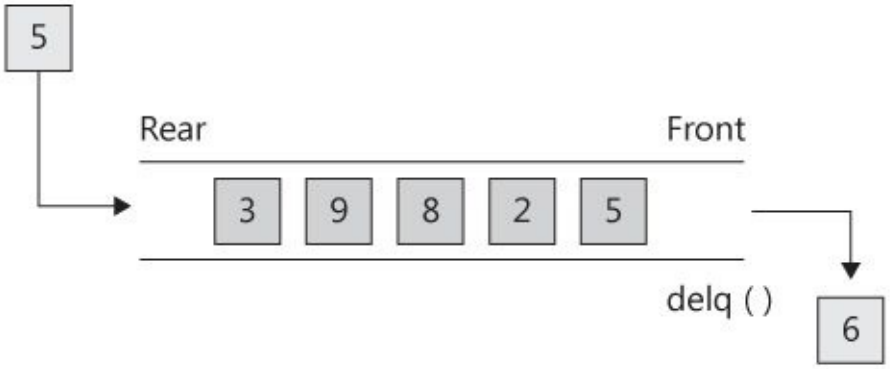
- There should be a base limit or criteria set so that it should not call itself.
- Every time whenever directly or indirectly the procedure calls itself it should reach near to the base.

If the procedure has these two properties then it is said to be a well defined recursive procedure.

Note: Divide-and –conquer algorithm which is a kind of binary search can be said to be a recursive procedure because it may be viewed as calling itself when the smaller sets are considered.

Queue

This follows FIFO that is first in first out just contrast from stack. In this insertion takes only in one end which is known as front and deletion takes place in other end that is known as rear.



Queue

Priority Queues

These are elements which assigned with some priority and when it comes to the action of deletion or something else the below rules are to be followed:

- Anything with more priority has to be processed before than that with less number of priorities.
- When two of them have same priority then they are processed in according to position they are placed in the queue.

Algorithm:

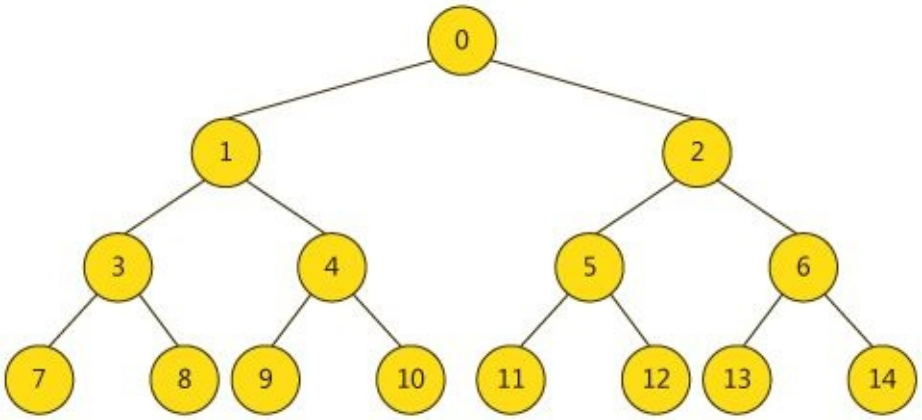
- Traverse the one way list until finding a node with priority number exceeding N. Insert the Item in front of node.
- If node exists or was discovered then insert as last element of the list.

Procedure for deleting and processing the first element in a priority queue:

- Find the smallest value such that FRONT [VALUE] ≠ NULL.
- Delete and process the front element in row VALUE of QUEUE.
- Exit.

Trees

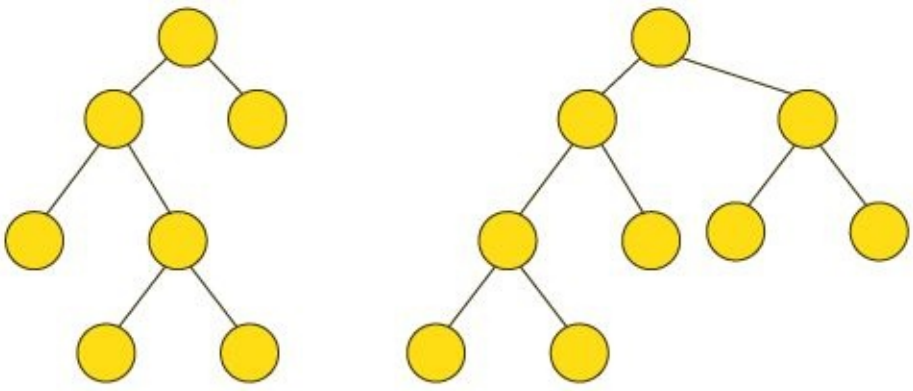
- Tree is defined as a set or a collection of nodes. If the tree is empty when there are no nodes available then the tree is stated as the null tree or an empty tree.
- Tree has a special node represented by R, which is the root node. The left side of the root node is left successor sub tree and the right one is said to be the right successor sub tree.



Tree

Terminology

- The height of the tree which is also stated as the depth of the tree is the total number of nodes present in the tree.
- The height of the root is equal to the height of the tree.
- A binary tree is said to be a full binary tree if the nodes present in it is exactly zero or two.
- A binary tree which is filled from right to left with possible exception at the bottom level is known as complete binary tree.



(Full Tree) (Complete Tree)

Use of the binary tree

- It represents the relationship between the data.

- It can be used to show a hierarchical structure of the data.
- It provides a very effective and time saving searching of data.

Traversing binary tree

Three ways to traverse a binary tree are as follows.

- Pre-order
- In-order
- Post-order

Pre-order

- Root is processed.
- Left sub tree is traversed with respect to the root.
- Right sub tree is traversed with respect to the root.

In-order

- Left sub tree with respect to root is traversed.
- Root is then processed.
- Now traverse the right sub tree with respect to the root in in-order.

Post-order

- Traverse the left sub tree with respect to the root in post–order.
- Now traverse the right sub tree with respect to the root in post order.
- Root is then processed.

Example:

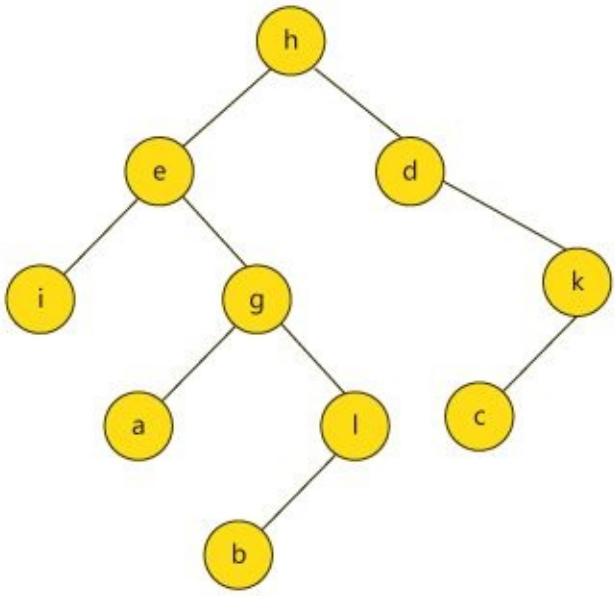
Provided a tree and four traversals:

Pre-order: h, e, i, g, a, l, b, d, k, c.

In-order: i, e, a, g, b, l, h, e, c, k.

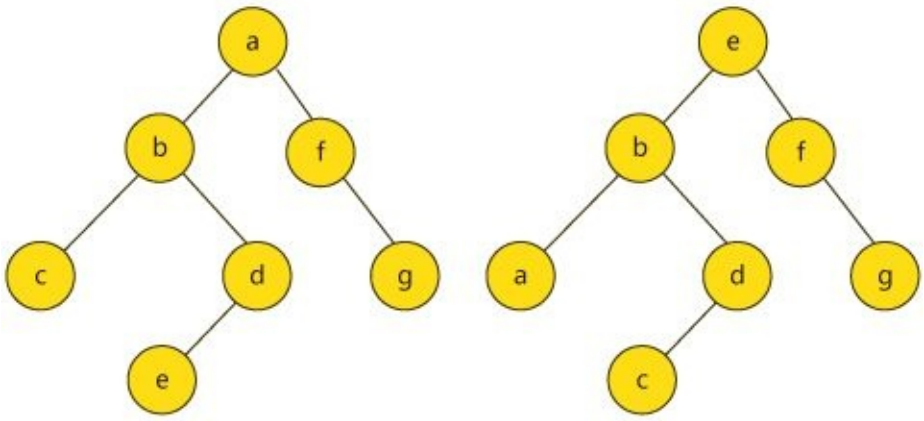
Post-order: i, a, b, l, g, f, c, k, d, h.

Level-order: h, e, d, h, g, k, a, l, c, b.

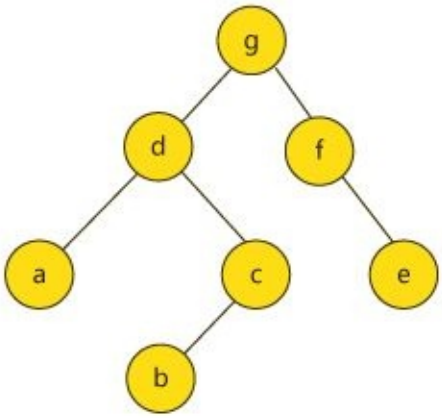


For the above example

Solution:



(Preorder) (Inorder)



(Postorder)

Binary search tree

- This particular data structure provides search action as its key character and that to in an average running time.

- It also provides the user to insert and delete or remove elements in the data structure.
- It is a contrast structure when compared to the sorted linear array and the linked list.
- It provides effective and sophisticated searching, insertion and removal of data.
- To support the binary search tree it is necessary that the data available in each node can be compared with each other.

Searching and insertion in binary search tree

- We have to compare the nodes of the tree with the VALUE we have.
- Now while comparing if we get VALUE< nodes, then we proceed to the left child of the node.
- But if we find that VALUE > nodes, proceed to the right side child of the node.
- We will repeat the above the above steps until we meet the node that is equal to the VALUE *i.e.* VALUE = node.
- Or we will meet an empty sub tree which symbolizes that we have reached the end and the search was unsuccessful.

Code

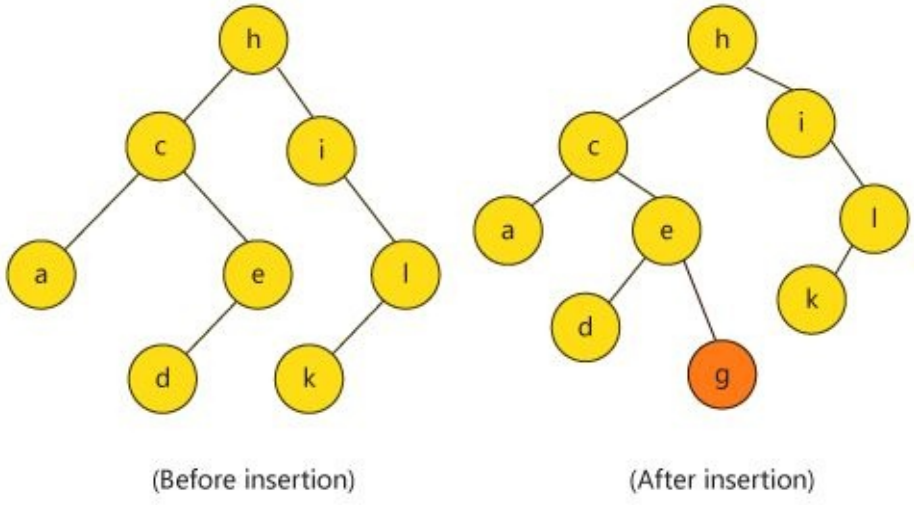
```
public class BINARYSTREE <VALUE extends Comparable<VALUE>>
{
private Node<VALUE> root;
private class Node<VALUE>
{
private VALUE data;
private Node<VALUE> left, right;
public Node(VALUE data)
{
left = right = null;
{
this.data = data;
}
}
```

……

}

Insertion in binary tree

- It is similar to the searching process. If the VALUE is already in existence in the tree then we don't insert the VALUE.
- New node will always replace the null reference area of node provided in the tree.
- For example you can refer the diagram in the next page.

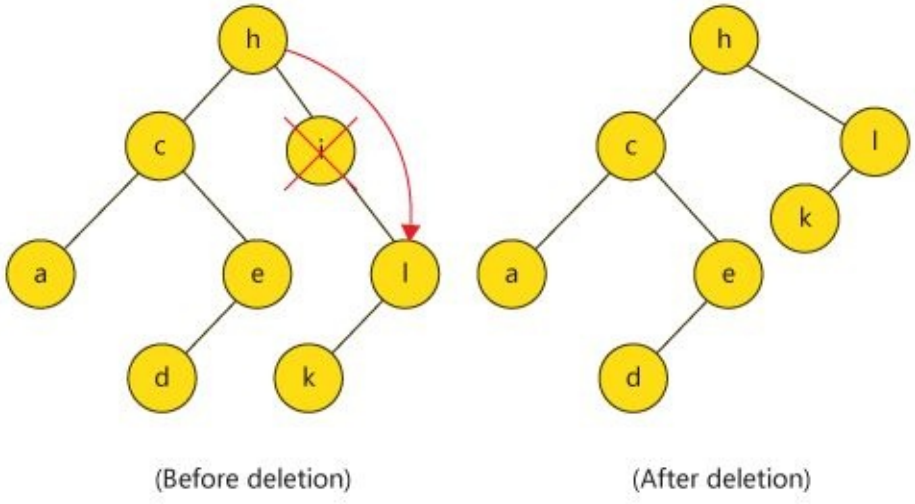


Insertion

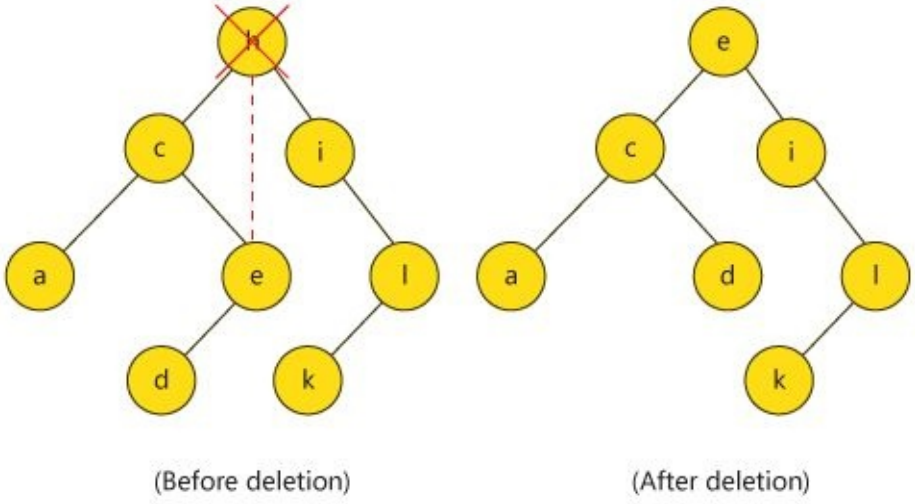
Deletion in binary search tree

Deletion is complex method in binary search tree as compared to that of the insertion method. Before we delete a node from the tree one has to consider the following conditions.

- Whether is it in a tree or not.
- Is it a leaf or not.
- Has only one child or not.
- Or has two children or not.



When a child node is deleted



When root node is deleted

When the root node is deleted then it checks the most eligible contender in the tree such as h had child node as *c* and *h*.

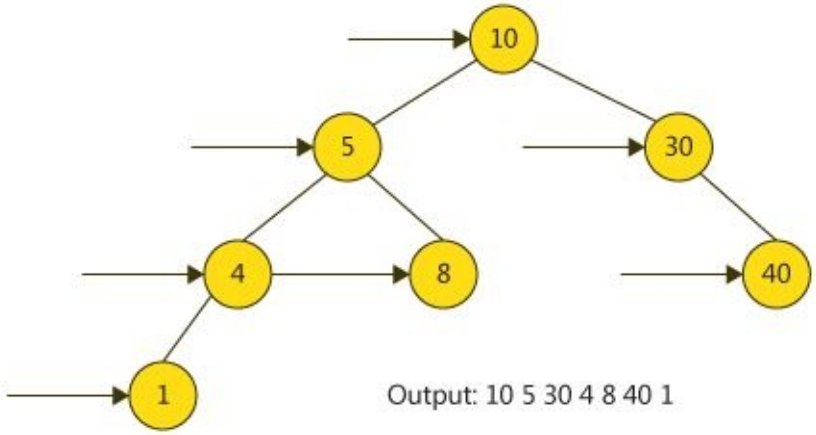
And according to the actual table only e in the whole tree satisfies the position of the root node since it is greater than c and less than *i*.

Therefore the after deletion of h, the position of *h I* swapped by *e*.

Note: Level order traversal is nothing but in which the nodes are processed level by level basis.

For example:

First the root node → then the child node → last the grandchild nodes.



Level order traversal

Graphs in Data Structures

- Graph is a collection of points that represent nodes of a tree which are interrelated with edges.
- Edges are nothing but a connection between nodes which is represented by an arrow *i.e.* “→ “.

Undirected and directed graph

When the edges have direction provided in the graph then it is called the directed graph and it is also known as digraph.

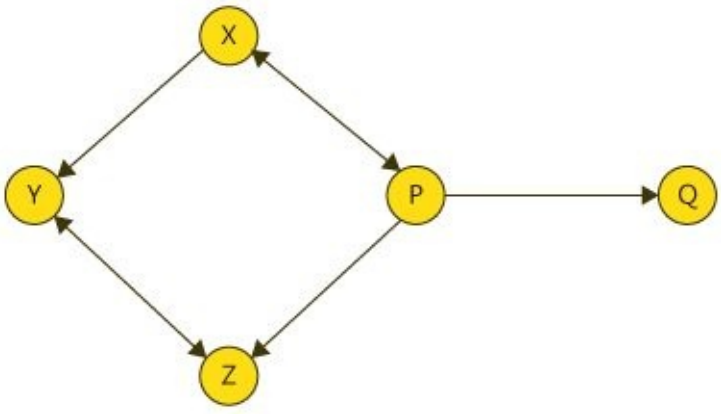
Suppose a directed graph with a directed edge $x = (a, b)$

Then x is also known as arc.

Therefore for this terms that are used are as follows:

- x emerges from a and dissolves in b.
- Now a becomes the initial point *i.e.* the origin of x.
- And b becomes the end or the destination point of x.
- Therefore in this case a will be the predecessor of b and b will be the successor of a, so both are adjacent to each other.

For an example:



Graph with vertex and nodes

Now,

G=<V, L>

Where, V= { X, Y, Z, P, Q }

L= { <X, B> <X ,P> <Y, Z> <Z, Y> <P, X> <P, Z> <P, Q> }

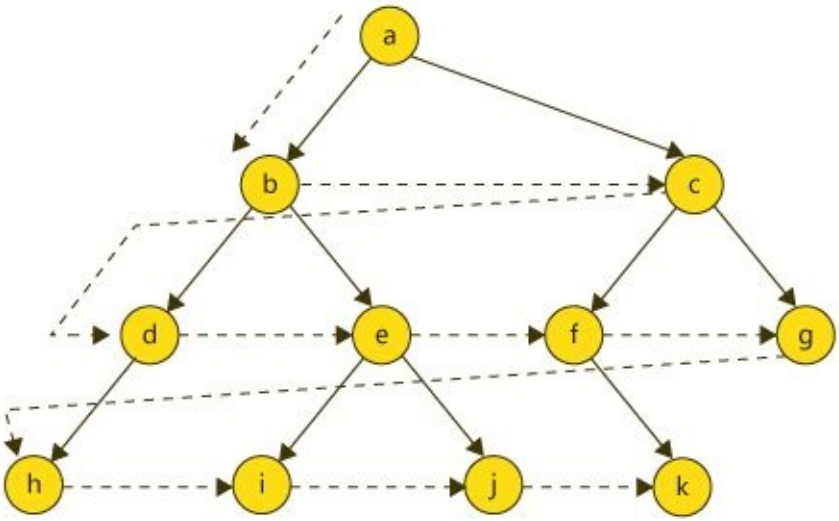
Traversing of graph

There are three states that are key in the execution of this process:

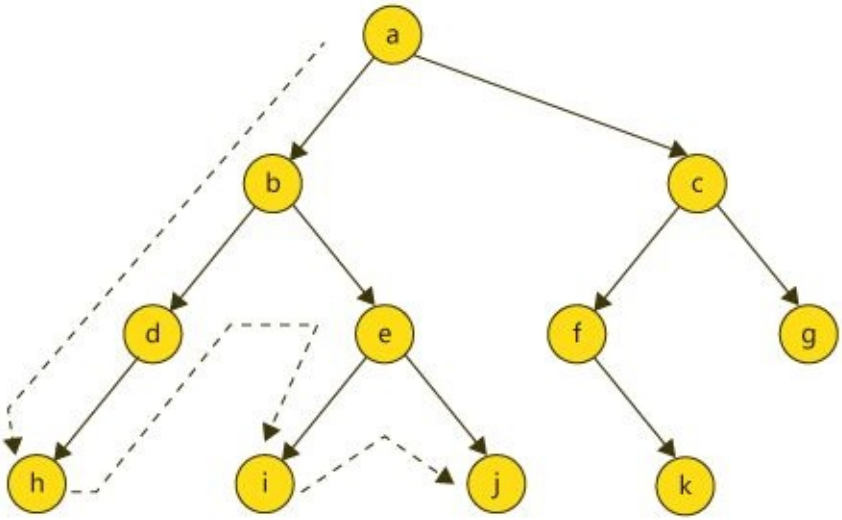
- Ready state which is the first or the initial state.
- Waiting state where nodes are in queue and are waiting to be processed.
- Third state is the processing state where nodes are processed.

There are two methodology of search in the graph:

- Breadth-first search
- Depth-first search



Breadth first search



Depth first search

Algorithm:

For BFS:

STEP 1: Submit all nodes to ready state.

STEP 2: Put first node in queue and the status changes to waiting state.

STEP 3: Repeat 4 and 5 until the queue is empty.

STEP 4: Process the first node and change then change the status as processed and then remove it from queue.

STEP 5: Add the rear node of the processed node and continue the process 4 again.

STEP 6: EXIT.

For DFS:

STEP 1: Submit all nodes to ready state.

STEP 2: Put first node in queue and the status changes to waiting state.

STEP 3: Repeat 4 and 5 until the queue is empty.

STEP 4: Process the first node and change then change the status as processed and then remove it from queue.

STEP 5: Add the rear node of the processed node and continue the process 4 again.

STEP 6: EXIT.

Note: Breadth-first search does the search in such a manner it would cover the elements in breadth wise first and after completion to last element in breadth manner it would go deep and then again it would do the search in the breadth manner and then go deep. Deep-first search does the search in such a manner it would cover the elements in depth wise first and after completion to last element in depth manner it would go in breadth.

Sort and search

Sorting and searching are some key operations in data structure with vast database.

Sorting

To sort is to arrange the data available in an ascending or increasing order.

For example:

$$1 < 2 < 3 < \ldots\ldots\ldots < n.$$

Complexity in sorting of data

- Comparing of data. For example: a < b or b > a.
- Switching of the contents.
- Assigning of the data. For example: b = a and then c =b or c=a.

Merging

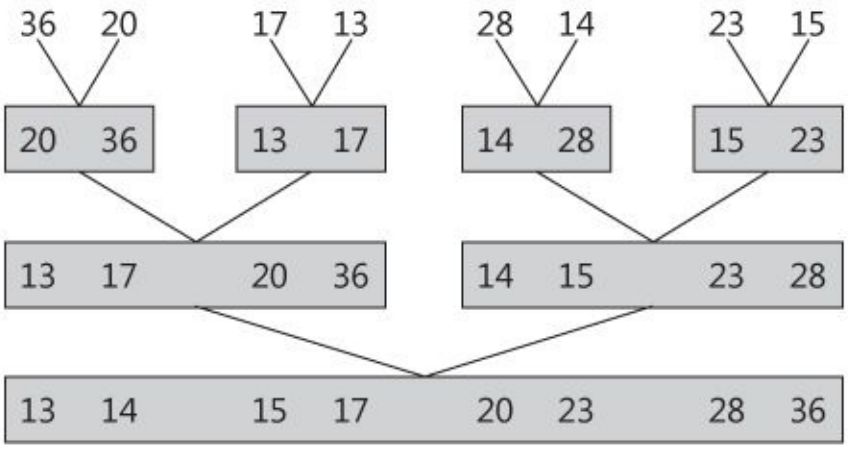
Merging can be defined as the process in which two data are merged to get a new data value.

For example:

A is a list which has been sorted with *x* elements. *B* is also a sorted list with *y* elements. Now to form a list *C* they merge element *x + y*, this is stated as merging.

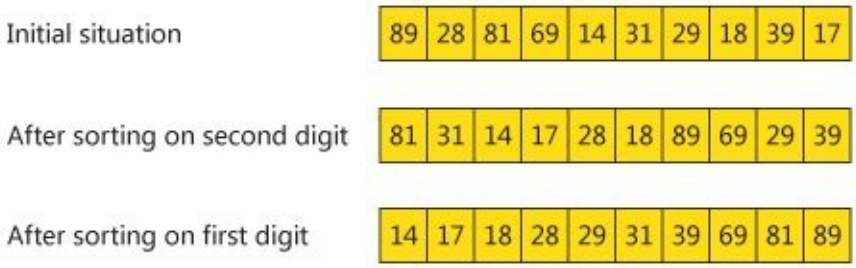
Two kind of sorts are been introduced in revised versions of data structure sorting methodology. Which are as follows:

- *Merge-sort:* In this the values are merged and then the particular elements in the array are sorted.



Merge-sort

- *Radix-sort:* It is similar method that is used by a card sorter.


Initial situation
89 28 81 69 14 31 29 18 39 17
After sorting on second digit
81 31 14 17 28 18 89 69 29 39
After sorting on first digit
14 17 18 28 29 31 39 69 81 89

Radix-sort

Searching

Searching techniques other than the sorted array, linked list and the binary tree which we haven't seen in the previous chapters is Hashing.

Hashing

- It's a technique in which the number of element is ignored to search i.e. there so need of knowing total number of element present in the data.
- To find the address of the data there as to be done some modification in the memory addresses *i.e.* form set of keys into a set of memory addresses.

Such function,

H: KEYS → MEMORY AD.

Therefore to perform hashing two of the procedure has to be carried out that is the.

- Hash function
- Collision resolution.

Hash function

Hash function is nothing the modification of the memory by providing a hash address to the element in the data. It is represented y H(k).

To evaluate a hash function following methods are been performed.

- Division method: In this the key is divided by the larger number(s) than the key(k) *i.e.*

H(k) =k(mod s)

- Midsquare method: In this the key(k) is squared *i.e.*

$H(k)= k^2$

- Folding method: In this k is folded or divided into number of parts *i.e.*

$H(k)=k_a + k_b +\ldots\ldots+ k_n$.

Collision resolution

The process in which there is a need to add a new record but the memory H(k) is already occupied. This situation is called collision.

Methods to overcome collision are called collision resolution. Methods are as follows:

Linear probing: This particular methods it carry outs a linear search for the record till it finds an empty location. Drawback of this probing is that it tends to face a problem *i.e.* clustering. This increases the search time for record. Double hashing and quadratic probing can lower down or overcome the clustering issue.

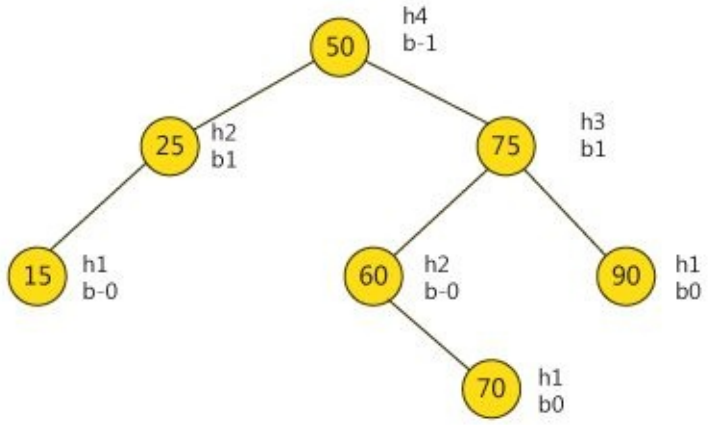
Chaining: In this the record is placed in the first position of the file and then it adds the recorder to the location that is empty pointed by the pointer.

AVL Search Trees

When a binary tree is empty it is considered as AVL tree. It is named after ***Adelson-Velskii*** and ***Landis*** therefore, it is known as AVL trees.

Representation of an AVL search tree

AVL is a linked representation of the node which has to show its balance factor therefore the number enclosed in bracket near the node represents its balance factor.



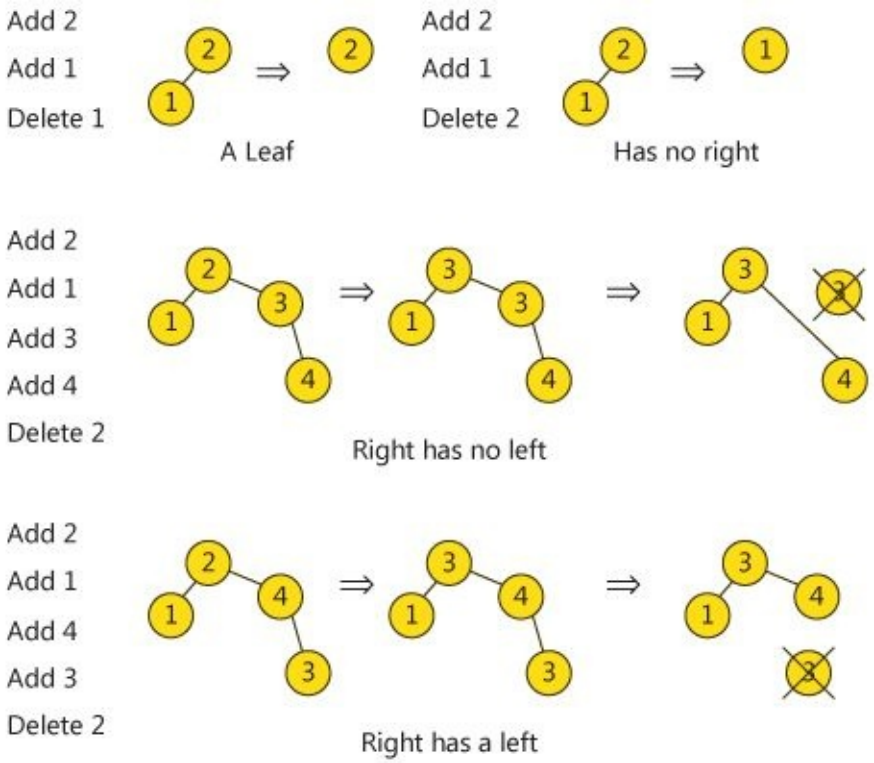
AVL tree representation

Insertion in an AVL search tree

- First it undergoes LL rotation *i.e.* insertion is done left sub of left sub tree of the node.
- Second is RR rotation.
- Third is LR rotation *i.e.* right sub of left sub tree of the node.
- Fourth is RL rotation *i.e.* left sub of right sub tree of the node.

Deletion in an AVL search tree

Deletion in AVL tree follows the similar procedure followed by the binary tree but in this the imbalance that will occur due to deletion is overcome by applying one or more rotation.



Deletion in AVL tree

For example:

When b =0 and the rotation is executed and then the next rotation executes when b=1.

Now when b=-1 the rotation will take place in the form of R-1. Therefore the procedure is

LL and r0 is same. LL and r1 are also identical but the outputs are different and LR and r-1 are identical.

Warshall's Algorithm

- This particular algorithm used to find the shortest path in the graph.
- This Algorithm is used to find the Boolean graded path matrix of a given graph.

Algorithm:

STEP1: Initialize P

If a [I,j]=0 then: set P[I, j] ≔ 0;

Else: set P [I,j] ≔ 1. // END OF LOOP//

STEP2: REPEAT 3 AND 4 for X = 1, 2,…, m: [UPDATE Q]

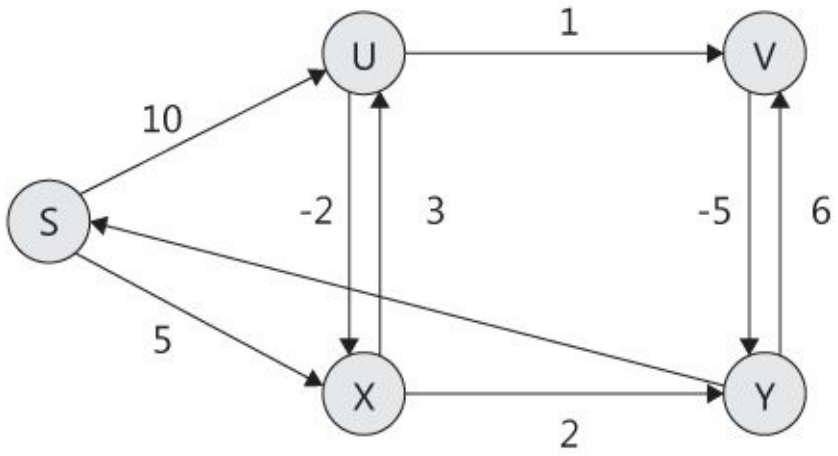
STEP3: REPEAT STEP 4 FOR I = 1, 2,…, m:

STEP4: REPEAT STEP 4 FOR J = 1, 2,…, m: set P[I,j]= P [I,j] ^{v}

(P [I, X] $^{\wedge}$ P[X,j]).

[END ALL RUNNING LOOPS]

STEP5: EXIT.



A Warshall's matrix weighted graph

Shortest path algorithm

Algorithm:

STEP1: Repeat fir a, b = 1, 2, …, M:

W [a, b] = 0, then: set [a, b] ≔ infinity;

Else: set Q [a, b] = W [a, b]

STEP2: Repeat steps 3 and 4 for X= 1, 2, …,m: [UPDATE Q]

STEP3: Repeat step 4 for a = 1, 2,…m.

STEP4: Repeat step 4 for b = 1, 2,…m.

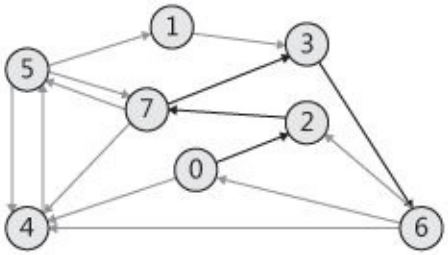
Set Q [a, b] = MIN (Q [a, b], Q [a, X] + Q [X, b]

[END OF ALL THE RUNNING LOOP]

STEP5: EXIT.

Edge-weighted diagraph

4->5	0.35
5->4	0.35
4->7	0.37
5->7	0.28
5->1	0.32
0->4	0.38
0->2	0.26
7->3	0.39
1->3	0.29
2->7	0.34
6->2	0.40
3->6	0.52
6->0	0.58
4->5	0.35
6->4	0.93



Shortest path from 0 to 6

0->2	0.26
2->7	0.34
7->3	0.39
3->6	0.52

Diagraph and shortest path

- Shortest path represents the smallest path calculated between the nodes.
- This algorithm is much more much more efficient than calculating the power of the given adjacent matrix.
- With the help of described graph we get know the path matrix that will help us to know the paths between the nodes is shortest or not.
- In first the Warshall's algorithm was used to describe the matrix but then was further modified and then was used to depict the shortest path.

Knowledge flow: more eBooks and Apps

- *Get more eBooks*
- *Get more apps from Google Play store*
- *Get more apps from Amazon app store*